**TÍCH HỢP LIÊN TỤC (CONTINUOUS INTEGRATION - CI) VÀ CHUYỂN GIAO LIÊN TỤC (CONTINUOUS DELIVERY - CD)**

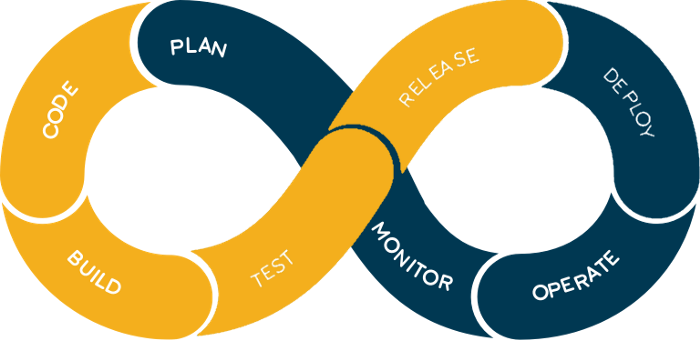
# **1. Giới thiệu về Tích hợp liên tục (Continuous intergation - CI) và Chuyển giao liên tục (Continuous delivery - CD)**

## **1.1 Tích hợp liên tục - CI**

### *1.1.1 Định nghĩa*

**Continous Integrations** (CI) - Tích hợp liên tục, là một phương pháp phát triển phần mềm mà yêu cầu những thành viên trong đội dự án tích hợp công việc của họ một cách thường xuyên, được khuyến nghị ít nhất 1 lần/ngày.

Mỗi lần tích hợp được tạo ra đều được kiểm thử và xác nhận tự động bởi một hệ thống CI. Hệ thống CI sẽ tự động phát hiện sự thay đổi về mã nguồn khi nhà phát triển đẩy các commit lên các nhánh của repository, sau đó tùy thuộc vào sự thay đổi mà người phát triển tạo ra liên quan đến việc tích hợp vào nhánh phát triển (push commit) hay tích hợp vào nhánh chính (merge request) mà hệ thống CI/CD sẽ lựa chọn hình thức kiểm thử tích hợp phù hợp. Nếu hệ thống CI/CD phát hiện lỗi xảy ra với mã nguồn trong quá trình kiểm thử tự động, hệ thống sẽ gửi thông báo tới người phát triển.



CI được các dự án triển khai theo kiểu Agile áp dụng phổ biến. Việc tự động hoá việc build, test và deploy có thể làm giảm bớt nhiều vấn đề đau đầu thường gặp trong các dự án nhằm mang lại hiệu suất làm việc tốt hơn, tránh lỗi và nâng cao chất lượng phần mềm.

### *1.1.2 Tác dụng*

- **Giảm thiểu rủi ro**: Dự án có càng nhiều người, sự tích hợp và triển khai càng nguy hiểm. Tùy thuộc vào từng vấn đề, việc sửa lỗi và giải quyết vấn đề có thể thực sự gây ra phiền phức và điều đó có thể có nghĩa là có nhiều thay đổi đối với code. Thực hiện chia nhỏ công việc, tích hợp hàng ngày hoặc thậm chí thường xuyên hơn có thể giúp giảm thiểu các loại vấn đề này ở mức tối thiểu, từ đó làm tăng chất lượng phẩn mềm.

- **Dễ dàng xử lý lỗi, nâng cao chất lượng code**: Mỗi lần tích hợp là một lần chạy auto test, build và deploy, nếu có phát sinh lỗi thì các thành viên đều nhận được thông báo và sửa ngay tức thì dựa trên những tiêu chí và cấu hình có sẵn (coding style, library, config, … ).

- **Giảm thiểu quy trình thủ công lặp đi lặp lại, tiết kiệm thời gian**: Trước đây, khi nhận được một push commit, hoặc merge request thì sẽ có một thành viên thực hiện pull code về và build, xem log server, … Công việc này khá tốn thời gian và lặp đi lặp lại gây nhàm chán cho người thực hiện. Ngược lại, khi CI được triển khai, những công việc đó được thực hiện một cách tự động. Thêm vào đó, hệ thống CI sẽ kết hợp với một số nền tảng khác (logging, tracing, testing, scanning, …) nhằm tối ưu hóa luồng thực hiện, giảm thiểu công đoạn lặp lại trước đây.

- **Tiết kiệm công sức cho đội kiểm thử**: Các test trong luồng CI được tự động hóa, nó có thể bao gồm kiểm thử đơn vị (unit test), kiểm thử tích hợp (integration test), … nên giảm được công sức cho tester.

### *1.1.3 Điều kiện triển khai*

Điều kiện cần thiết để triển khai CI cho dự án là một **hệ thống kiểm soát phiên bản** (Version Control System – VCS) như Github, Gitlab, SVN, Bitbucket, … và một **CI tool**. Việc có một CI tool là một điều không bắt buộc nhưng được khuyến nghị sử dụng để triển khai một cách đơn giản và dễ dàng hơn. Hiện nay có rất nhiều CI tool được cộng đồng sử dụng như GitlabCI, Jenkins, CircleCI, … (More about CI tool here <https://code-maze.com/top-8-continuous-integration-tools/> )

Ngoài những yêu cầu về công cụ thiết yếu, một số những tiêu chí cần thiết sau cũng là yếu tố để thực hành CI:

- **Duy trì một nhánh chính duy nhất trên repository, hạn chế phân nhánh quá nhiều**: Theo khuyến nghị của chuyên gia, việc tích hợp được tiến hành từ nhánh phát triển vào nhánh chính ít nhất một lần một ngày, vì thế nên việc có nhiều nhánh chính hoặc là phân nhánh phát triển quá phức tạp cũng dẫn đến khó khăn trong việc tích hợp.

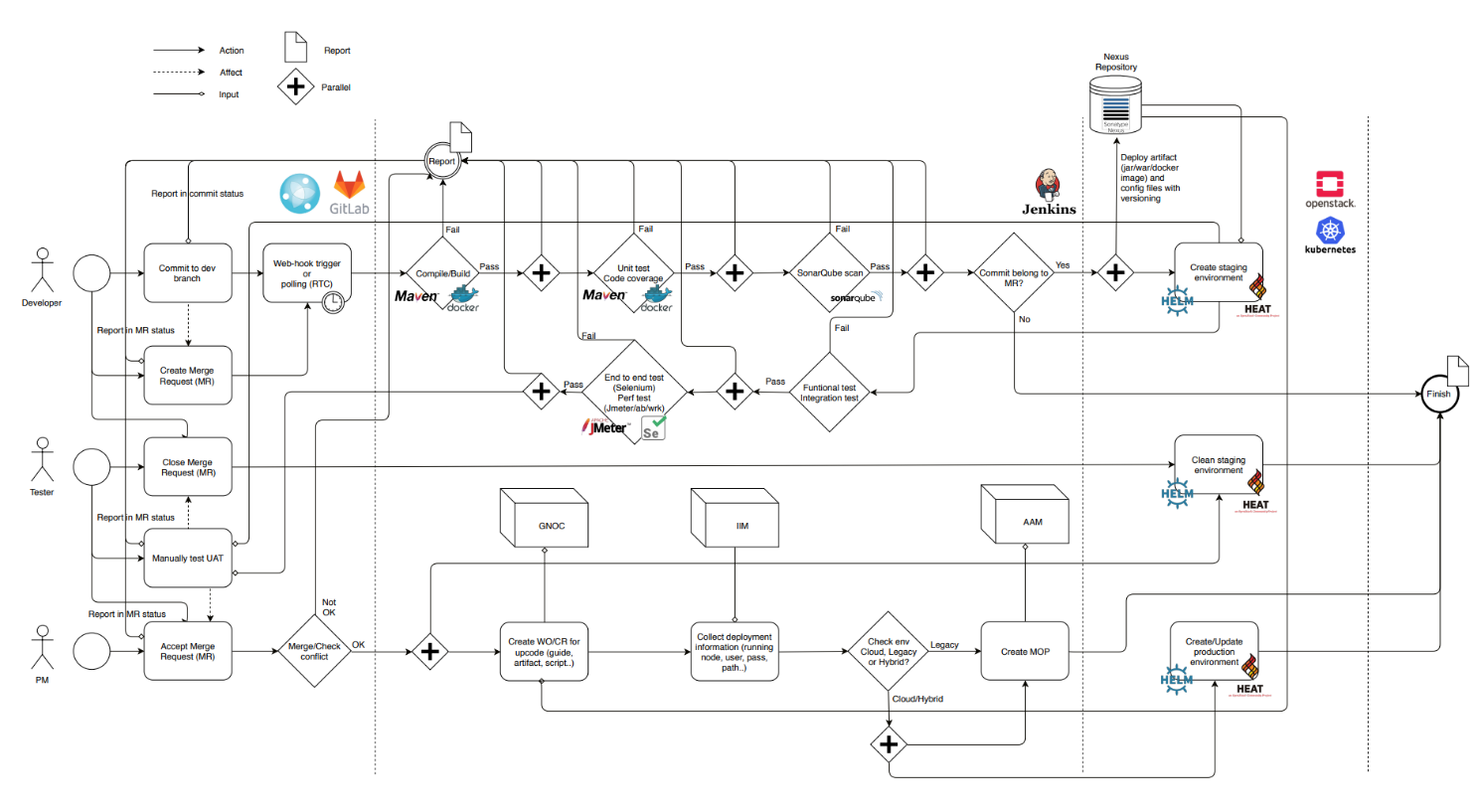
- **Developer cần tích hợp code thường xuyên và liên tục với kích thước vừa phải**: Code vừa phải tương đương với việc commit code thường xuyên sẽ làm cho việc xử lý vấn đề phát sinh tốt hơn, quản lý tốt hơn trong việc tích hợp.

- **Đảm bảo quá trình kiểm thử diễn ra nhanh, trả về kết quả kiểm thử trong thời gian ngắn**: Điều này giúp cho các developer có thể phát hiện vấn đề và sửa lỗi nhanh chóng. Để tối ưu hóa các bài kiểm thử, unit test sẽ được ưu tiên thực hiện trước vì nó nhanh và tốn ít thời gian, còn kiểm thử tích hợp (integration test) và kiểm thử hệ thống (system test) sẽ được thực hiện và triển khai sau.

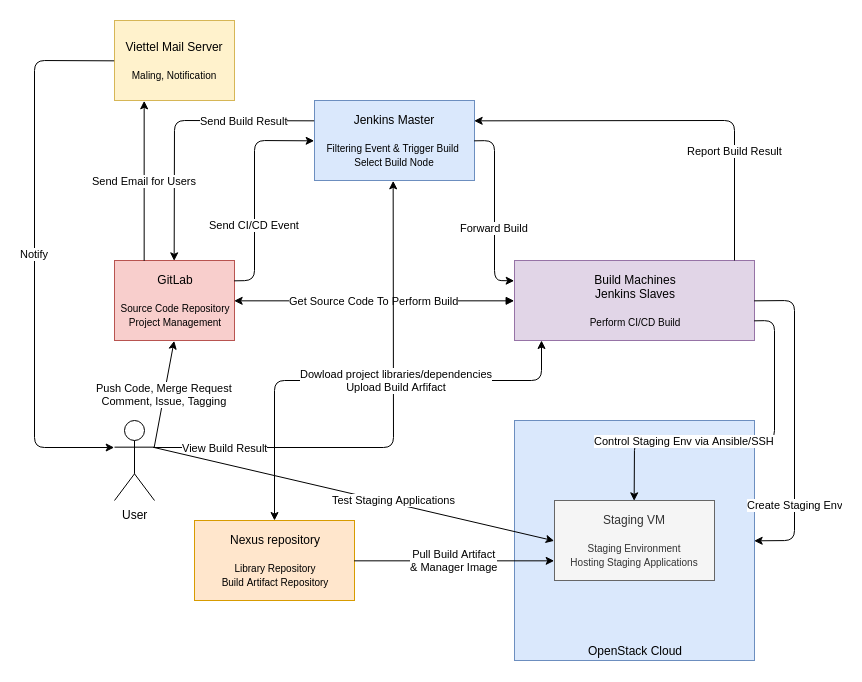
### *1.1.4 Luồng hoạt động*

Trong mục này, một số luồng CI tiêu chuẩn được mang ra làm mẫu, có thể học hỏi và triển khai trong các dự án của công ty.

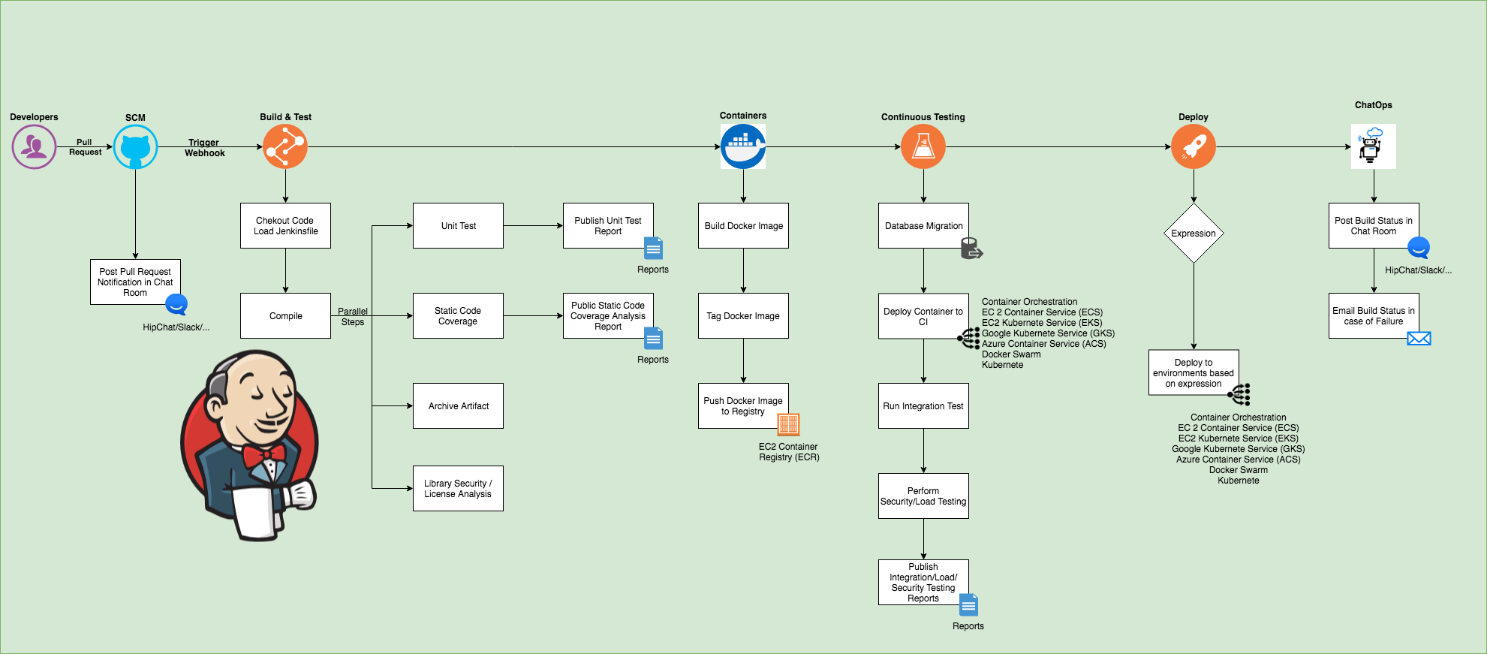
- Luồng CI/CD tiêu chuẩn của Viettel ban hành:

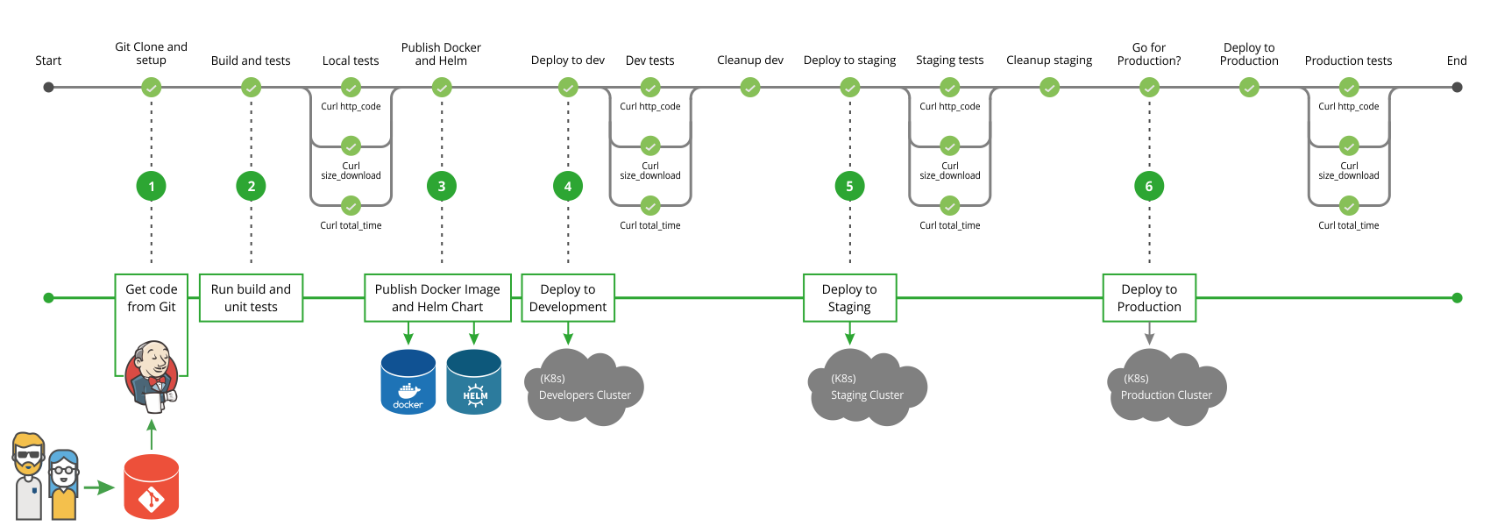


- Luồng CI/CD được áp dụng cho dự án Vsmart của Viettel:



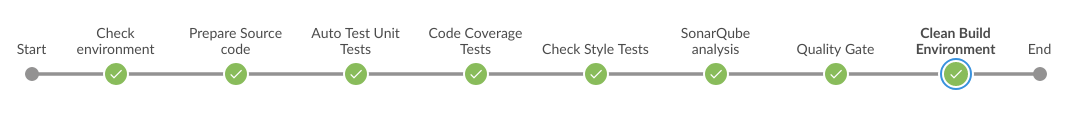
- Một vài ví dụ khác về luồng CI:



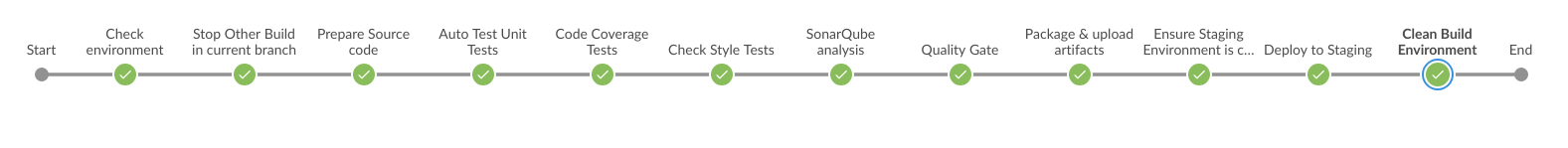


- Mỗi sự kiện tích hợp (push commit, merge request, create tag, …) có thể áp dụng những luồng CI khác nhau tùy vào mục đích sử dụng.

Ví dụ luồng xử lý push commit:

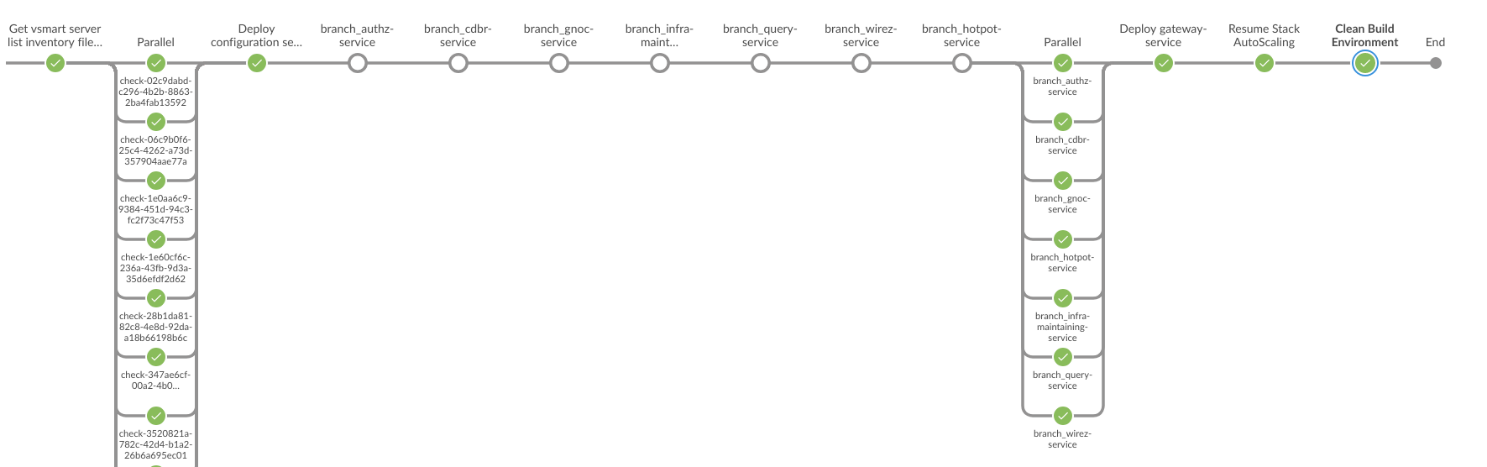


Ví dụ luồng xử lý merge request:



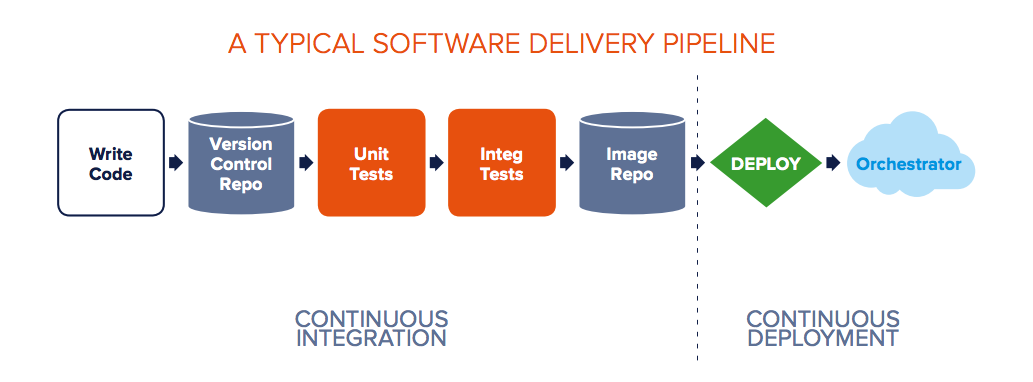
Mỗi luồng xử lý có các node nối tiếp nhau, mỗi node xử lý một công việc cụ thể như chạy unit test, check out source code, quét SonarQube, … Các thao tác cụ thể của từng node được định nghĩa trong file cấu hình.

Ngoài ra, để tiết kiệm thời gian chạy một luồng CI/CD, các node có thể được thực hiện một cách song song như sơ đồ mẫu dưới đây:



## **1.2 Chuyển giao liên tục - CD**

**Continuous delivery (CD)** – Chuyển giao liên tục, được coi là một bước phát triển của CI, hoàn tất giai đoạn chuyển giao tới người dùng trên các môi trường khác nhau (môi trường thử nghiệm – stagging, hoặc môi trường sản phẩm – production).



Vì nó là một bước phát triển của CI nên các giai đoạn thực hiện trong CD cũng tiếp nối từ CI. Nếu CI là tự động pull code, tự động quét SonarQube, tự động unit test và đóng gói thì CD sẽ thực hiện các công việc như tự động thực hiện kiểm thử tích hợp, kiểm thử hệ thống, tự động build code lên server (lên môi trường stagging hoặc production), tự động đẩy gói đã đóng lên trên repository chung của công ty, …

# **2. Triển khai thử nghiệm với Gitlab CI**

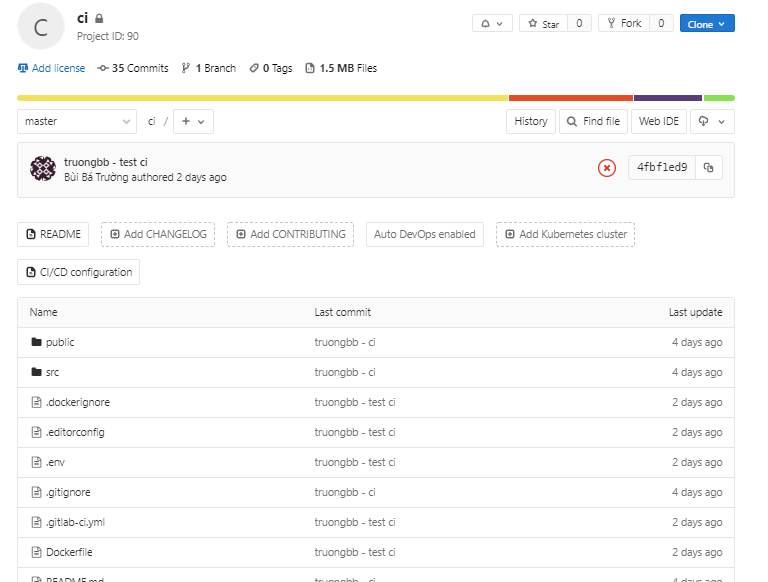
Cơ chế hoạt động của GitlabCI đó chính là có một bộ phận sẵn sàng lắng nghe các sự kiện từ các member trong project của Git (push commit, merge request, create tag, comment, ….), khi có sự kiện nổ ra, nó sẽ kết hợp cùng với file cấu hình CI được viết sẵn trong project (sẽ đề cập sau) để thực hiện các công việc mà người cấu hình luồng CI mong muốn. Bộ phận này được gọi là một **Gitlab runner**.

Vì vậy, chúng ta cần có một **Gitlab runner** sẵn sàng lắng nghe và nắm bắt sự kiện cùng với một **file cấu hình** chứa nội dung thực hiện để tiến hành luồng CI. Gitlab runner trong tài liệu phần triển khai này được cài đặt ngay trên remote SSH server mà ta mong muốn deploy sản phẩm (ngay trên centOS server).

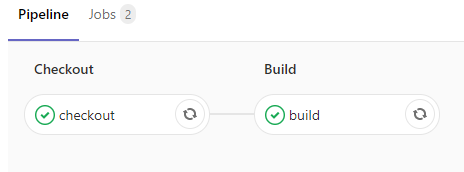
## **2.1 Triển khai luồng CI với project ReactJS**

Trong tài liệu này, VCS và CI tool được sử dụng lần lượt là Gitlab và GitlabCI để thực hiện deploy một project lên một remote SSH server, cụ thể là CentOS server.

Trước hết, ta cần có một repository trên Gitlab, trong tài liệu sử dụng một project Reactjs đơn giản nhất để thực hiện demo.

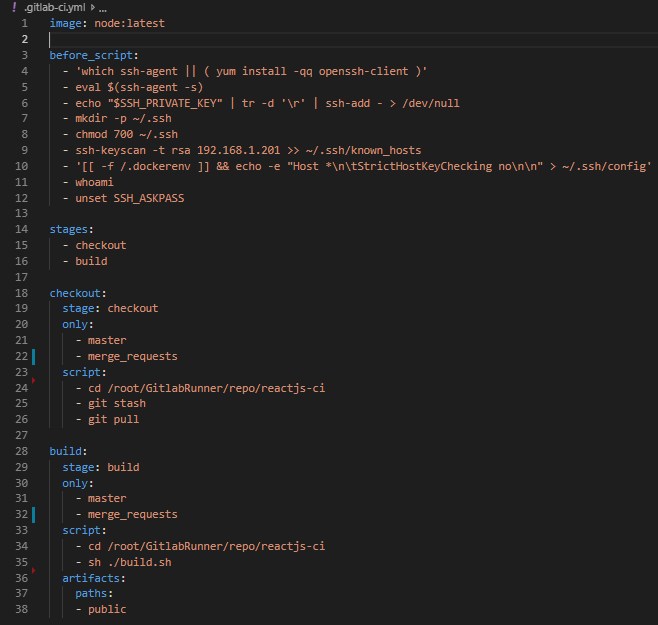


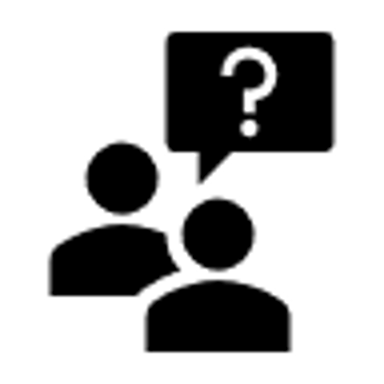
Luồng CI dự kiến cho các project front-end nói chung được triển khai đơn giản bao gồm 2 bước: Checkout code từ Gitlab và build lên server mong muốn



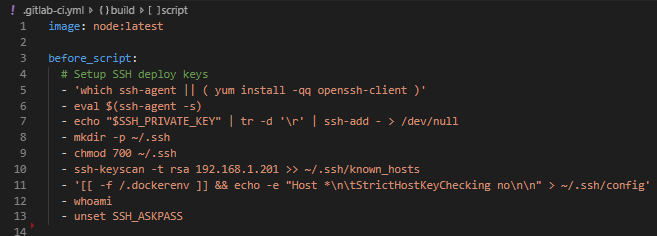
### *2.1.1 Tạo file cấu hình*

Trước hết, ta cần tạo một file cấu hình có tên ***.gitlab-ci.yml***, bên trong chứa nội dung cần cấu hình thực hiện luồng CI. Cụ thể, file cấu hình trong phần demo của tài liệu chạy các lệnh linux đơn thuần (shell script) nên có dạng như sau:

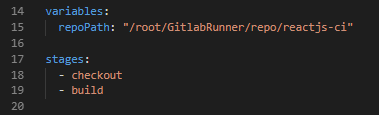


**** Giải thích file cấu hình**

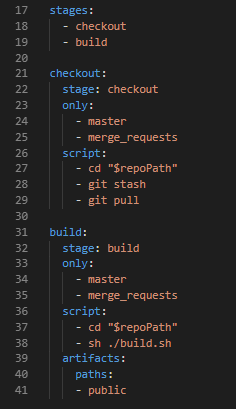
Đầu tiên, vì đây là project reactjs nên ta định nghĩa image dùng để build project và đoạn mã để kết nối tới sever dựa vào một ***SSH\_PRIVATE\_KEY*** đã được tạo từ trước (cách tạo key sẽ được đề cập trong phần cấu hình Gitlab runner ở bên dưới):



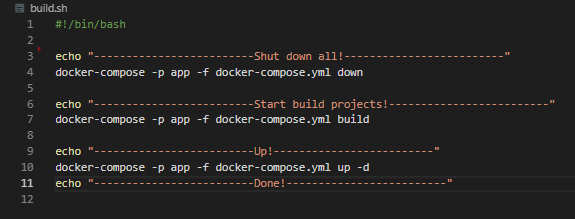
Tiếp theo là liệt kê các biến toàn cục được dùng trong file cấu hình và các bước/giai đoạn sẽ được thực hiện trong pipeline (gọi là ***job*** hoặc ***stage***):



Trong từng giai đoạn thực hiện, ta sẽ cấu hình nhánh được trigger stage, đoạn mã cần thực thi, …



Ví dụ như trong stage build, ta chỉ thực hiện khi có một push commit lên nhánh master và build image artifact sẽ lưu ở đường dẫn ./public của project. Đoạn script được thực hiện có tác dụng clone project từ git về và chạy các lệnh trong file ***build.sh.*** File build.sh này cũng chứa các lệnh để build và deploy project sử dụng docker.



Sau khi có file cấu hình này và push lên Gitlab, Gitlab sẽ nhận biết được và kích hoạt luồng CI đầu tiên. Nhưng luồng CI này sẽ ở trạng thái chờ (***pending***) do chưa có bộ phận thực thi (runner). Bước tiếp theo là thực hiện cài đặt và đăng ký runner.

### *2.1.2 Cài đặt và đăng ký runner*

Các bước cài đặt và vận hành một Gitlab runner được nêu rõ ràng ở [tài liệu của Gitlab](https://docs.gitlab.com/runner/). Cụ thể, muốn cài đặt Gitlab runner trên centOS ta cần làm các bước sau đây (tất cả các lệnh linux được viết dưới đây dành cho centOS/Fedora/RHEL):

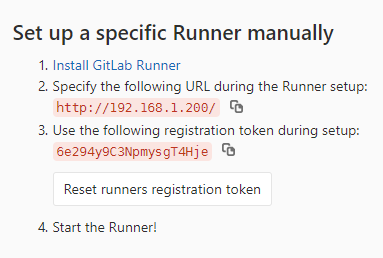
* Thêm repository:

curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.rpm.sh | sudo bash

* Cài đặt phiên bản mới nhất (nếu muốn cài phiên bản cũ hơn vui lòng tra tài liệu trong bên trên):

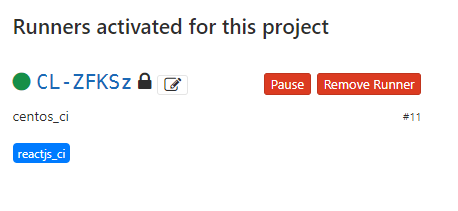
sudo yum install gitlab-runner

* Đăng ký Gitlab runner cho CentOS server: Chỉ cần chạy lệnh sudo gitlab-runner register. Trong quá trình đăng ký, chương trình sẽ hỏi URL và token cho runner, 2 thành phần này được lấy trên mục Setting của project trên Gitlab (***Setting >> CI/CD >> Runner >> Expand >>*** Mục ***Set up a specific Runner manually***).

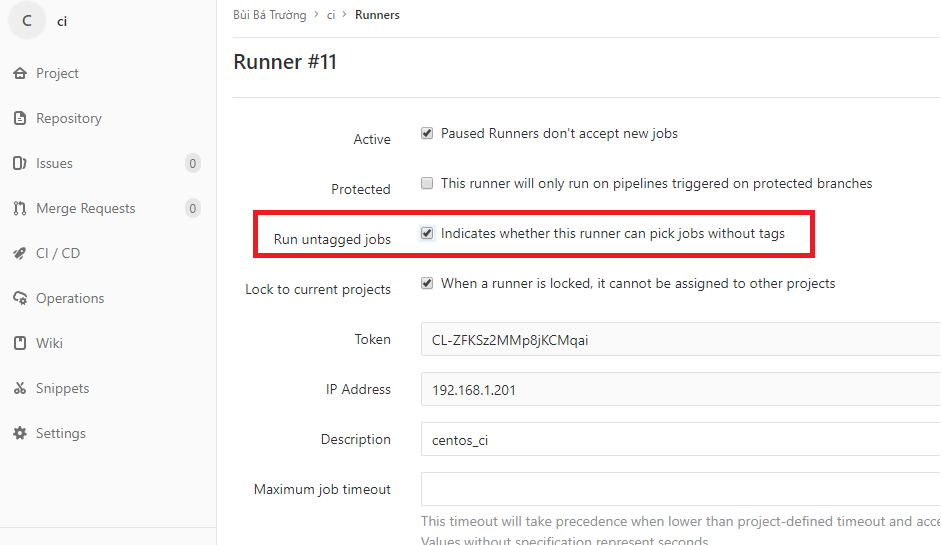


Ngoài ra, ở bước cuối cùng của phần đăng ký, ta cần chọn thành phần thực thi cho runner (***runner executor***) tùy theo mong muốn thực hiện (docker, shell, virtualbox, ….), cụ thể trong demo của tài liệu này chọn là ***shell*** (thực hiện câu lệnh linux đơn thuần).

Sau khi đăng ký thành công, ta cần sử dụng lệnh gitlab-runner start để khởi chạy runner vừa tạo, trạng thái runner trên Gitlab web sẽ chuyển thành đang chạy.



Lưu ý, chúng ta cần tích vào mục ***Run untagged jobs*** trong phần edit của runner trên Gitlab để runner này có thể lắng nghe cả những sự kiện mà không có tag.



* Bước tiếp theo là tạo một ***SSH key pair*** trên CentOS server để gắn kết Gitlab runner với repository của mình ở trên Gitlab

Trước hết, ta cần tạo ra một cặp SSH key được mã hóa bằng cách sử dụng lệnh:

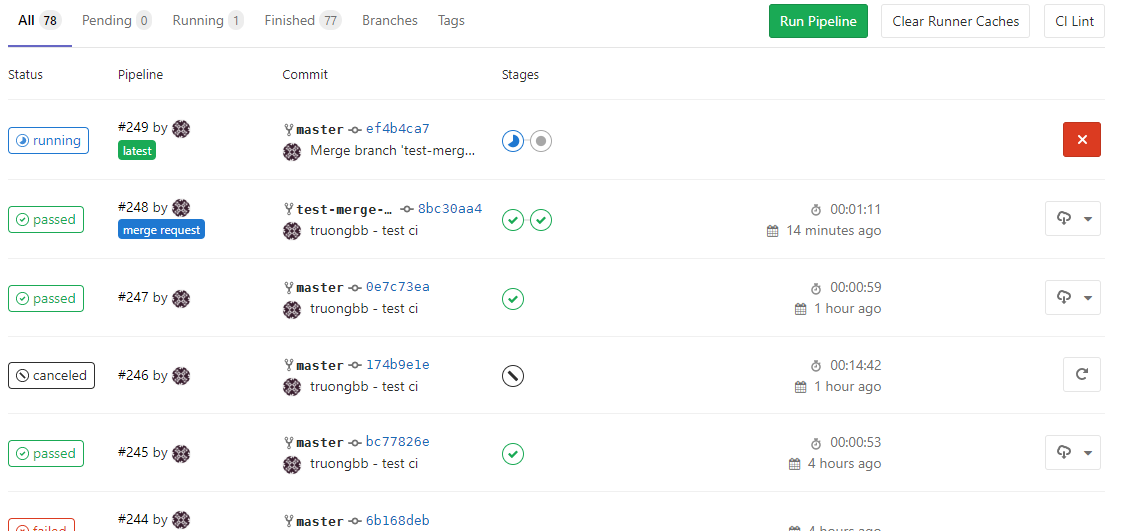
ssh-keygen -o -t rsa -b 4096 -C "email@example.com"

với email của của người tạo. Trong quá trình tạo SSH key, ta sẽ được hỏi nơi lưu, passpharase cho key.

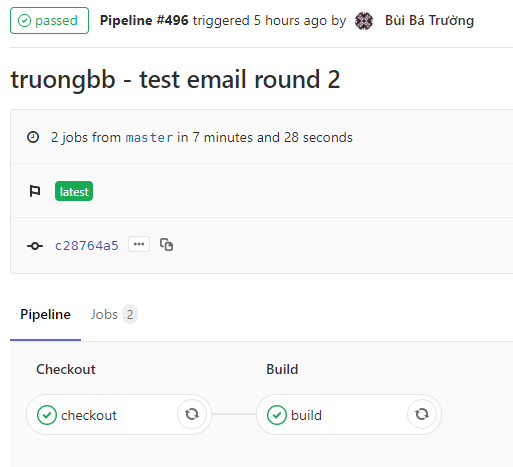
Sau khi tạo xong, ta nhận được 2 file có tên giống nhau, nhưng một file có phần mở rộng là ***.pub*** chứa public key, file còn lại không có phần mở rộng chứa private key. Ta copy private key, sau đó tạo một biến môi trường cho project trên Gitlab bằng cách vào mục ***Setting >> CI/CD >> Variables***, điền giá trị trường Key là ***SSH\_PRIVATE\_KEY*** (bắt buộc) và giá trị trường Value là private key mà ta vừa copy, sau đó lưu lại.

Một điểm lưu ý khác chính là nếu runner đã khởi chạy và bạn đã có file cấu hình mà các luồng CI (CI pipeline) luôn luôn ở trạng thái chờ (pending) thì bạn cần chạy thêm lệnh gitlab-runner run để khắc phục điều này.

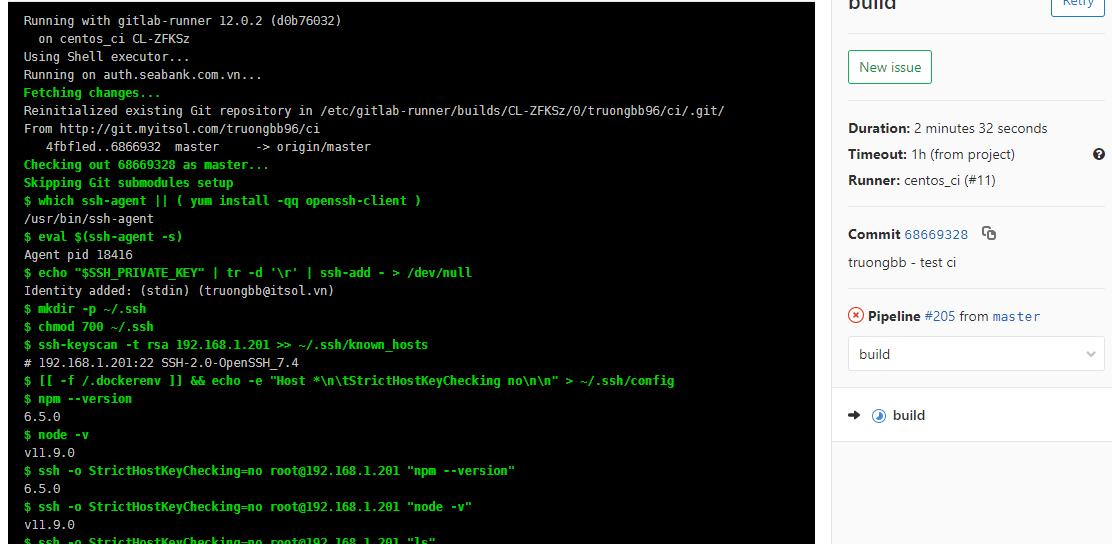
Sau khi đã cài đặt và đăng ký runner thành công, luồng CI đang ở trạng thái pending bên trên sẽ tự động chuyển sang running và được runner tiến hành thực thi.



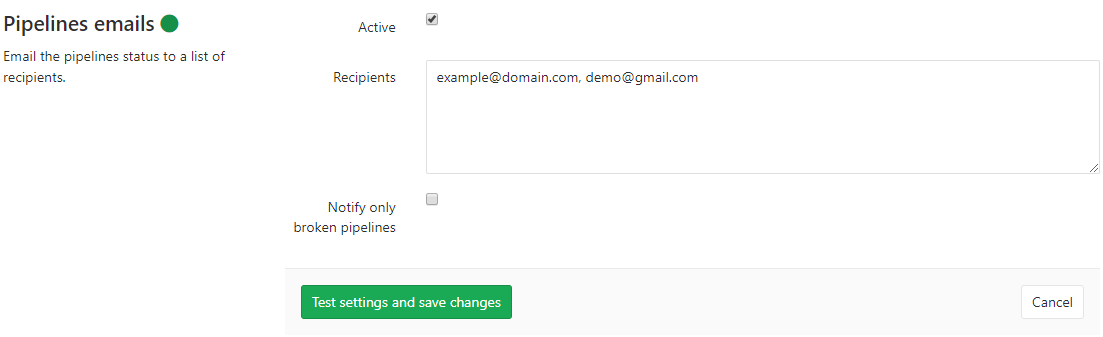
Khi bấm vào chi tiết từng pipeline, ta sẽ thấy luồng thực hiện của pipeline đó:



Chi tiết về nội dung và kết quả của các bước thực hiện có thể xem cụ thể ở log của từng job:

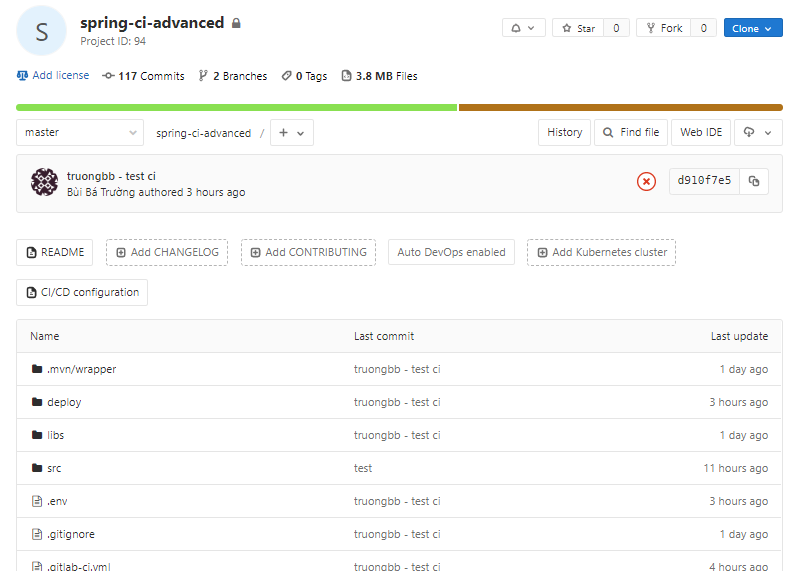


Ngoài ra, để biết được khi có một luồng CI pipeline thực hiện xong, ta sẽ cấu hình việc gửi email tới những thành viên trong đội dự án bằng cách vào mục ***Setting >> Integrations >> Project Services >> Pipelines emails***. Trong mục này ta cần active service này lên và điền các email mà ta muốn gửi tới sau khi thực hiện luồng CI.

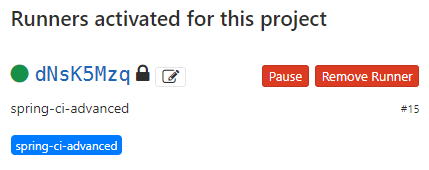


## **2.2 Triển khai CI/CD với project Spring boot Maven**

Chúng ta cũng sẽ có một project Spring boot Maven trên gitlab để chuẩn bị thực hiện cài đặt luồng CI/CD.



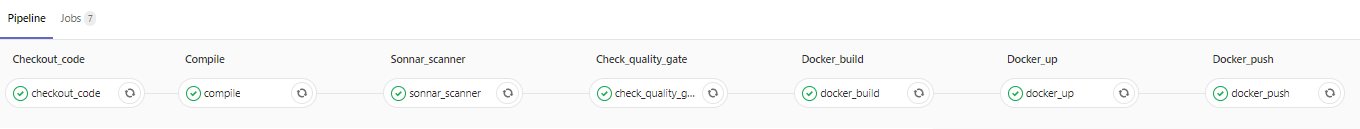
Ngoài ra, dựa vào các bước ở phần trên, chúng ta tạo cho project này một runner để lắng nghe sự kiện.



Với project backend như thế này, luồng CI và CD được chia tách riêng biệt. Cụ thể, luồng CI sẽ được trigger khi có một push commit lên trển gitlab. CI pipeline sẽ thực hiện pull code từ Gitlab về, sau đó biên dịch và thực hiện quét SonnarQube.

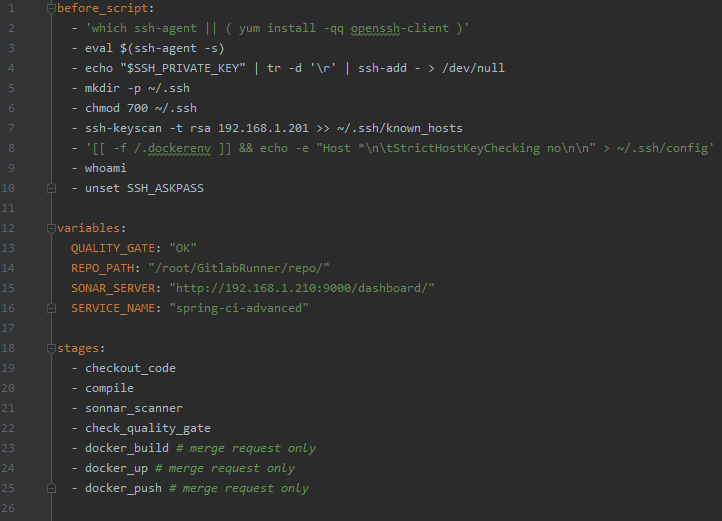


Ở luồng CD, bước build sẽ được thực hiện, trong bước build sẽ bao gồm tạo file docker image, build lên SSH remote server và push docker image lên Nexus repository.



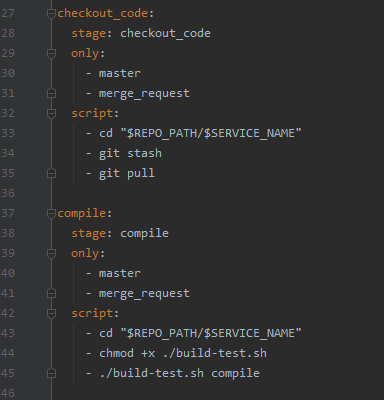
Và chúng ta vẫn sẽ cấu hình gửi email kết quả sau khi thực hiện xong luồng CI như cấu hình đã đề cập phần trên.

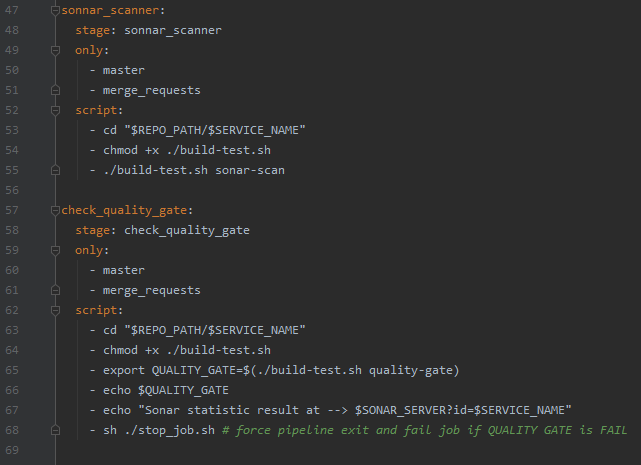
File cấu hình của 2 luồng này đều có phần chung là kết nối tới remote server SSH, định nghĩa các biến toàn cục và các stage sẽ được thực hiện trong luồng.



### *2.2.1 Cấu hình luồng CI*

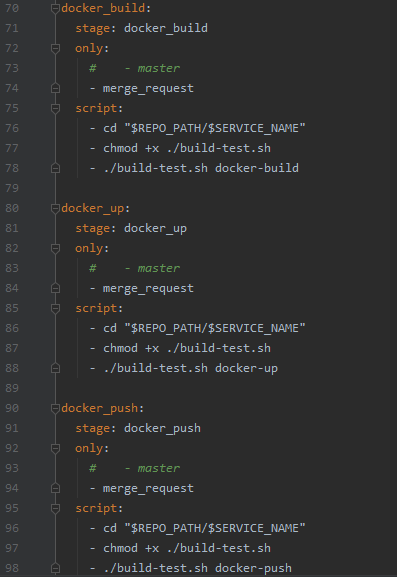
Trong luồng CI, các bước lấy code từ Gitlab, biên dịch, quét SonarQube và lấy kết quả được thực hiện. Luồng CI này được trigger khi có một push commit lên Gitlab hoặc một Merge request được chấp thuận (accept).





### *2.2.2 Cấu hình luồng CD*

Luồng CD được phát triển tiếp từ luồng CI sẵn có, ngoài các bước của luồng CI, luồng CD thực hiện build, up service và đẩy docker image nhận được từ bước build lên trên Nexus repository.



### *2.2.3 File* ***build-test.sh***

Trong các bước thực hiện luồng CI/CD của project này, chúng ta thấy các luồng đều thực hiện các sub-script được gọi từ file ***build-test.sh***. File này là một file định nghĩa các lệnh docker, shell script thực hiện cho một hành động (như build, up, push, sonar scanner, …)

